Solution to Bristols Cipher for the University Cipher Challenge

By Students of the Univ. of Birmingham

This document describes how we cracked Bristol's Cipher for the University Cipher Challenge. This work was carried out by students of the University of Birmingham Hacking Club, PhD students and students on our M.Sc. in Computer Security. All the files mentioned in this document should be found in an accompanying zip file.

Stage 1: enc.txt to enc.c

The cipher came in the form of a text file containing hex, enc. txt. We wrote this hex to a file as bytes and spotted that most of it was ascii:

```
2f30 2244 2475 6f6c 6b6e 2a7a 722d 717e  /0"D$uolkn*zr-q~
7476 3279 0307 367c 067c 0c14 0c11 070e  tv2y..6|.|......
0e41 4d43 0913 0910 1811 0f1d 1912 1c23  .AMC...........#
5051 523c 3d55 3f40 4143 4345 6b6c 5e7d  PQR<=U?@ACCEkl^}
....
```

We tried many different encodings to turn this ascii into something we could read and in the end we found that removing 0 from the first byte, 1 from the second, and so on (modulo 128) turn the hex into a readable C file. We did this using a purpose written python program txt2c.py and got back a C program:

```
$ python txt2c.py enc.txt
// A piece of code for encryption + encipherment

// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
// This is the entry from the University of Bristol

// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

#include <stdio.h>

#include <stdlib.h>
```

We called the resulting file enc.c, compiling it and running it we got the following string as output:

```
69B475AAC07D91056CA7DCD1130231FB272BFC44B76C4D7BE35F56980A6F839B5
63AB14EAA24C561A1A7D4937E51C76480594C8898B2BC1BF5EC8EDBDAC091B199
```

Stage 2: Whitespace in enc.c

The first thing we noticed about the C code was that it had a lot of unnecessary blank lines. We also found a lot of spaces and tabs at the end of the lines (this was particularly clear when the file was viewed in a text editor). We tried the "snow" white space steganography tool, with no success, so we looked for a pattern ourselves. We found that if space was replaced by a 0 and tab by a 1, then the last 8 pieces of white space on each line encoded to ascii. We wrote a python program to decode this for us (whiteSpaceBinAscii.py), and this program returned another hex string:

```
$ python whiteSpaceBinAscii.py enc.c
741EA6E06769748DEA85D88B6BE91E545E58A8C16D23B4932696415BE5C217D7E
0111D9DAAC43302FEF07000A7A461CD70FD994FDC99F1C15565B0B254EB1E9DBD
```

Stage 3: A Hidden Program

We solved the first 2 stages quickly but the 3rd stage took much longer. Of course, we tried reversing the enc.c program to "decode" the string from the whitespace and we tried many different ways of combining the 2 strings, however nothing produced any readable text. So we when back to the C program and studied it in detail.

While analysing the code we found many sections that looked like they have been written just to include some particular symbols, e.g. <> + − [].,

This reminded us of the language "Brainfuck" (BF)[1], which uses only these symbols and treats all other symbols as comments.

Stripping out all non-BF characters we found that we had a valid BF program that took in 130 characters and returned a string. We wrote a c equivalent program (encbf.c) and ran it on the output of the original C program to get a third hex string:

```
$ gcc -o encbf encbf.c
$ echo 69B475AAC07D91056CA7DCD1130231FB272BFC44B76C4D7BE35F56980
A6F839B563AB14EAA24C561A1A7D4937E51C76480594C8898B2BC1BF5EC8EDBD
AC091B199 | ./encbf

5C8AA423C97A80FAA64B777A01845A8E593B31F7BF6F909BE5A837B18EC3rm enF023D
B0BDFA7728998047F23CBB3A987C9DF9584ACA8310B3AB5CEB31E5ABB58CA4E56
```

Stage 4: The Solution

We then tried running the BF program on a lot of different strings and tried a lot of different ways of combining the hex strings we had found.

In the end, we found that XORing each set of 2 hex symbols, from each of the 3 hex strings, gave as an ascii message and solved the challenge. This function is performed by our last program solution.py:

```
$ python solution.py
A winner is you! Here is a random string to prove you won: 35sEbr
```

Acknowledgment

Many University of Birmingham students contributed to this solution, but particular credit must go to students on the M.Sc. in Computer Security and students from the University of Birmingham Hacking Club.

[1] http://en.wikipedia.org/wiki/Brainfuck